Разработка ОС реального времени для цифрового сигнального процессора

В. В. Рубанов, К. А. Власов

Аннотация. В статье рассматривается задача построения операционной системы реального времени (ОСРВ) для цифрового сигнального процессора. Описывается конкретная реализация ОСРВ MicroDSP-RTOS, разработанная в ИСП РАН. Рассматривается архитектура ОС и предоставляемые функции. Кроме того, описываются доработки в инструментарии кросс-разработки MetaDSP для обеспечения эффективной разработки и отладки многозадачных приложений под платформу MicroDSP-RTOS.

1. Введение

В настоящее время всё большую роль начинают играть встраиваемые системы на основе цифровых процессоров обработки сигналов (ЦПОС). ЦПОС используются практически во всех областях деятельности человека — в быту, науке, медицине. Важнейшим программным компонентом, лежащим в основе функционирования таких систем, является операционная система, которая позволяет запускать одновременно несколько разных программ и организовывать взаимодействие между ними для решения одной общей задачи. Для встраиваемых систем обработки сигналов характерны *операционные системы реального времени* (ОСРВ). Эти системы применяются в тех случаях, когда главная задача — успеть среагировать на событие в рамках строго определенного максимального времени реакции. Например, это может быть сигнал на датчике, отображающем текущее состояние какого-то объекта в реальном времени. Возможна ситуация, когда состояние объекта на короткое время меняется, а потом возвращается обратно, и если это изменение останется незамеченным и необработанным системой, последствия могут быть самыми разными — от совершенно безобидных до катастрофических.

Важно также отметить, что возможность «успеть среагировать на событие» вовсе не означает высокую скорость работы. Система может работать относительно медленно, и всё же являться системой реального времени. Главное отличие ОСРВ от ОС общего назначения — это некий фиксированный промежуток времени, в течение которого система *гарантированно* среагирует на событие и выполнит его обработку. Величина

этого промежутка времени определяется решаемой задачей и является одним из требований к разрабатываемой системе. Он может быть очень коротким, но может быть и длинным, важно лишь то, что он фиксирован и известен заранее.

Применение систем реального времени может быть самым разнообразным. Рассмотрим, например, работу сотового телефона. Его процессор должен выполнять одновременно довольно много задач: приём и кодирование речи при разговоре, отправку закодированного звука на ретрансляционную станцию, приём входящего закодированного звукового потока, раскодирование и воспроизведение его; плюс к этому необходимо обмениваться со станцией всякого рода служебной информацией — такой как переход из зоны в зону и переключение на другую станцию, отслеживание уровня сигнала, при необходимости — усиление его и так далее. Причём многие из этих задач должны выполняться в реальном времени, без задержек. Например, задержка в обработке сигнала с микрофона приведёт к тому, что часть фразы будет утеряна; запаздывание с переключением на другую ретрансляционную станцию может привести к потере связи и разрыву соединения. Таким образом, применение операционной системы реального времени в данной ситуации не только оправдано, но и необходимо.

В данной статье мы рассмотрим операционную систему реального времени, разработанную в ИСП РАН для частной «системы на чипе» (System-On-Chip) на базе цифрового сигнального процессора MicroDSP 1.1, когда на одном общем кристалле размещаются сам процессор, модули расширения, программная память и два банка памяти данных. Размещение их на одном кристалле позволяет обеспечить очень быстрый доступ к ячейкам памяти (обращение к памяти занимает один такт). Размер банков памяти данных может меняться от 0 до 65536 16-битных слов; они независимы, и к ним можно обращаться одновременно. Программная память может составлять до 256К слов (4 страницы по 64К слова), размер слова составляет 24 бита (длина инструкций процессора). Стек организуется программно, при помощи трёх специальных регистров, содержащих границы стека и текущее положение указателя стека. Процессор поддерживает до 15 программируемых прерываний с индивидуальной настройкой приоритетов и маскированием, а также доступны три таймера.

Предполагалось, что система будет работать одновременно не более, чем с 64 задачами. Каждая задача имеет свой статический приоритет, причём двух задач с одинаковыми приоритетами быть не может. Планировщик задач выбирает для запуска задачу с наивысшим приоритетом из тех, что находятся в состоянии готовности (то есть, в принципе, допустима ситуация, когда какая-то задача ни разу не получит управления). Процессорное время выделяется задачам квантами, длительность кванта может варьироваться. Увеличение длительности кванта ухудшает параллелизм, но снижает затраты, связанные с переключением процессов; уменьшение длительности, соответственно, – наоборот. Для каждой задачи оптимальное значение

длительности кванта будет своим, поэтому возможность настраивать длительность кванта времени весьма полезна. Также ОС должна предоставлять базовые функции по управлению процессами и реализацию основных примитивов синхронизации и межзадачного взаимодействия.

В данной статье мы рассмотрим функциональность разработанной системы и её возможности. В разделе 2 будет описана собственно сама операционная система и предоставляемые ей функции. Раздел 3 описывает доработки в интерфейсе интегрированной среды и отладчика MetaDSP, позволяющие создавать и отлаживать многозадачные приложения.

2. ОСРВ MicroDSP-RTOS

Операционная система MicroDSP-RTOS предназначена для работы с многозадачными приложениями. Под задачей мы в данной статье подразумеваем составную часть какого-либо сложного приложения, работающую самостоятельно и практически независимо от остальных частей. Также задачи часто называют процессами или потоками. Операционная система производит необходимые действия по распределению процессорного времени между задачами и обеспечивает переключение между ними, сохраняя и восстанавливая контексты так, что переключение остаётся для задач совершенно прозрачным. Также система содержит набор функций для выполнения различных служебных действий (функции ядра ОС), таких как инициализация внутренних структур, запуск самой ОС (по сути — запуск аппаратного таймера), механизм сохранения/восстановления контекста и переключения задач и так далее. Что касается предоставляемого системой API, здесь присутствуют функции для управления самими задачами, их состоянием, функции для синхронизации и взаимодействия задач между собой, для управления динамической памятью. Рассмотрим возможности системы более подробно.

2.1. Общая функциональность

Всего в системе может присутствовать максимум 63 пользовательских задачи. Сами задачи являются обычными функциями без параметров, чаще всего представляющими собой бесконечный цикл. Каждой задаче соответствует приоритет от 0 до 62, задаваемый при подключении, причём не может существовать двух задач с одинаковыми приоритетами. Системное время квантуется, и в каждый квант времени выполняется задача, имеющая наивысший приоритет (самый высокий приоритет соответствует значению 0) среди тех, которые не находятся в состоянии ожидания. На приведённой ниже схеме (Рис. 1) можно видеть основные состояния, в которых может находиться задача, и возможные переходы между состояниями.

Рис.1. Схема переключения состояний задач

После истечения каждого кванта времени (system tick) вызывается функция обработки прерывания таймера. Эта функция выполняет следующие действия:

- обновляет значение времени ожидания (таймаута) для каждой задачи, находящейся в состоянии ожидания по какой-либо причине;
- если у какой-то из задач таймаут истёк, переводит эту задачу в состояние готовности (**Ready**);
- после этого из всех задач, находящихся в состоянии готовности, выбирает задачу с наивысшим приоритетом и переключается на неё (сохранив контекст текущей задачи, если это требуется).

В системе всегда присутствует одна внутренняя задача, называющаяся **background** и имеющая самый низкий возможный приоритет — 63. Это значение ниже приоритета любой из пользовательских задач, и поэтому эта задача выполняется только тогда, когда все пользовательские задачи находятся в состоянии ожидания; таким образом, задача **background** является индикатором простоя системы. В начальной реализации эта задача представляла собой цикл, состоящий из нескольких инструкций NOP. В дальнейшем туда была добавлена инструкция IDLE, которая останавливает процессор до тех пор, пока не появится запрос на прерывание. Тем самым было снижено энергопотребление процессора на время простоя.

2.2. Управление задачами

1. **Подключение задачи.**
   Задачи подключаются динамически, поэтому в начале нужно явно вызвать функцию подключения для каждой задачи, которая будет выполняться.

При подключении задача добавляется во внутренние структуры данных системы, стек инициализируется стартовым контекстом, который будет восстановлен при первом переключении на эту задачу.

2. **Отключение задачи.**
   Эта функция полностью удаляет задачу из всех внутренних структур данных операционной системы, и дальнейшая работа с этой задачей становится невозможной. Для повторного использования задачи её требуется снова подключить, после чего выполнение задачи пойдёт с самого начала.
3. **Приостановка выполнения задачи.**
   Выполнение задачи может быть на время приостановлено. Это может потребоваться, например, чтобы запустить выполнение менее приоритетной задачи, имея более приоритетную активную задачу. При вызове функции указывается длительность задержки в квантах времени. По истечении этого времени задача переводится в состояние готовности и, если она является наиболее приоритетной, получает управление.
4. **Восстановление из приостановленного состояния.**
   Эта функция позволяет при необходимости досрочно поставить приостановленную задачу в очередь на выполнение, переведя её в состояние готовности.
5. **Блокировка задачи.**
   Блокировка задачи очень похожа на отключение. Единственное отличие состоит в том, что при блокировке состояние задачи полностью запоминается и может быть восстановлено путем вызова соответствующей функции, после чего задача продолжит выполнение с того же места, где была остановлена. В случае же отключения продолжить выполнение задачи нельзя, её можно только подключить заново, и она начнёт выполняться с самого начала.
6. **Вывод из режима блокировки.**
   Эта функция выполняет действие, обратное блокированию. Состояние задачи восстанавливается в то, которое было до блокировки, за одним только исключением: если задача выполнялась, она переводится в состояние готовности, а не выполнения. Таймаут (для состояний приостановки и ожидания) начинает уменьшаться с каждым квантом времени; задача может получать сигналы и сообщения и так далее.
7. **Получение данных, переданных задаче.**
   При подключении задачи ей можно передать какие-либо данные (один из параметров функции подключения — адрес произвольной структуры данных). Данная функция позволяет получить этот адрес, чтобы иметь возможность обратиться к переданным данным.
8. **Изменение приоритета задачи.**
   Данная функция позволяет изменить приоритет любой задачи (при этом новый приоритет не должен быть занят). Для удобства работы, чтобы не нужно было запоминать и определять, каким приоритетом обладает каждая

задача в каждый момент времени, обращение к задачам производится через уникальные идентификаторы. По сути, эти идентификаторы являются переменными, содержащими реальные значения приоритетов задач. В результате к задаче, скажем, номер 3 всегда можно обращаться через переменную TASK3, не задумываясь о том, менялся ли у неё приоритет, и если менялся, то какой он сейчас, поскольку эта переменная всегда будет содержать корректное текущее значение приоритета.

2.3. Синхронизация и взаимодействие задач

Для синхронизации задач и взаимодействия их друг с другом предусмотрены следующие механизмы:

- сигналы;
- семафоры;
- сообщения;
- очереди сообщений.

Сигналы.

Для уведомления о наступлении какого-либо события задача может послать другой задаче сигнал. Рассмотрим, например, ситуацию, когда задача должна считать данные из порта ввода-вывода. В этом случае можно перевести задачу в режим ожидания сигнала. Когда данные будут готовы, другая задача или процедура обработки прерывания пошлёт соответствующий сигнал, в результате чего первая задача будет активирована и сможет выполнить чтение данных.

Семафоры.

В MiscoDSP-RTOS реализованы двоичные семафоры и семафоры со счётчиком. Как те, так и другие обычно используются для обеспечения контроля работы с общими ресурсами.

Двоичный семафор может принимать значения 1 или 0, что означает, соответственно, доступность и недоступность ресурса. Если задаче требуется доступ к ресурсу, она должна вызвать системную функцию, которая проверяет, доступен ли ресурс. Если он уже занят, то задача переводится в состояние ожидания, а управление передаётся следующей наиболее приоритетной задаче. Если же ресурс доступен (значение семафора равно 1), то он блокируется (значение устанавливается в 0), и управление возвращается в задачу. Когда работа с ресурсом завершена, его нужно освободить вызовом соответствующей функции. При этом из списка всех задач, ожидающих данный ресурс, выбирается наиболее приоритетная и переводится в состояние готовности (а если она имеет более высокий приоритет, чем текущая задача, то происходит переключение). Если таких задач нет, то ресурс просто помечается как свободный (значение устанавливается в 1), и управление возвращается в выполняющуюся задачу.

Отличие семафоров со счётчиком от нескольких двоичных семафоров состоит только в том, что ресурсы могут использоваться несколькими задачами одновременно. Например, если есть канал передачи данных, состоящий из четырёх параллельных линий, работающих независимо, то для работы с ним может использоваться семафор со счётчиком, изначально равным 4. Когда задаче требуется линия передачи данных, она делает системный запрос. В результате, если количество доступных линий больше нуля, оно уменьшается на 1; в противном случае задача переводится в режим ожидания до тех пор, пока одна из задач, блокирующих ресурсы, не освободит используемую линию. Стоит отметить, что данный подход существенно отличается от использования нескольких двоичных семафоров. Семафор со счётчиком не делает различия между контролируемыми им ресурсами. В вышеприведённом примере задача, ожидающая ресурс, переводится в состояние готовности при освобождении *любой* из четырёх линий. При использовании же четырёх двоичных семафоров пришлось бы ждать освобождения какой-то одной конкретной линии, даже если все остальные уже свободны.

Сообщения.

Сообщения позволяют задачам обмениваться данными. Этот механизм может использоваться, когда одной задаче требуется не только известить другую о наступлении события, но и передать ей какие-то дополнительные сведения (например, уведомление о завершении чтения из порта с указанием адреса буфера, где находятся считанные данные). Принцип действия является таким же, как у сигналов и семафоров: задача, которой требуются данные, вызывает системную функцию, которая определяет, доступны ли данные в запрошенном почтовом ящике. Если данные доступны, то они передаются задаче, и ей возвращается управление. Если же ящик пуст, то задача переводится в режим ожидания. При посылке сообщения проверяется, нет ли задачи, ожидающей получение данных из этого же почтового ящика. Если такая задача есть, она переводится в режим готовности, и при необходимости выполняется переключение задач, если же данные никем не запрошены, они будут храниться в почтовом ящике до первого запроса.

Очереди сообщений.

Очереди сообщений могут использоваться, когда данные поступают нерегулярно, и неизвестно, будет ли задача успевать сразу считывать все поступающие сообщения. Очередь сообщений организована в виде циклического буфера типа FIFO. Длина очереди, а также количество очередей сообщений, которое может использоваться в программе, задаётся в конфигурационных файлах RTOS-проекта. Работа с очередями организована таким же образом, как и с одиночными сообщениями, с тем лишь различием, что можно хранить не одно сообщение, а несколько.

2.4. Работа с динамической памятью

В операционной системе MicroDSP-RTOS реализованы простейшие механизмы работы с динамической памятью. Разумеется, механизмы, используемые во многих современных системах программирования, не подходят для ОС реального времени, т. к. время их выполнения недетерминировано и зависит от фрагментации памяти. Поэтому для данной ОС был разработан свой подход, позволяющий избежать фрагментации памяти и сделать время выполнения функций детерминированным.

В конфигурационных файлах RTOS-проекта задаётся общее количество динамической памяти, которое будет использоваться программой. Пользователь должен оценить, сколько потребуется памяти, учитывая расход памяти на служебную информацию. Динамическая память организована в виде набора областей (пулов) памяти, каждая из которых состоит из некоторого количества буферов одинакового размера. Для работы необходимо предварительно создать пул, указав количество буферов в нём и их размер (разумеется, суммарный размер не должен превышать количество свободной динамической памяти), после чего системными вызовами можно выделять буфера и возвращать их обратно в пул, помечая их, тем самым, как свободные. При такой функциональности дефрагментировать память нет необходимости, поскольку работа ведётся только с буферами одинакового размера.

2.5. Использование процедур обработки прерываний

Из присутствующих в процессоре линий прерывания одна занята самой операционной системой (прерывание таймера для реализации многозадачности). В своей программе разработчик может реализовывать обработку других прерываний, но при этом необходимо иметь в виду некоторые особенности системы.

В первую очередь, это возможность запрещать прерывания на некоторое время. Это бывает необходимо в случае, когда программе требуется исключительный доступ к данным, и никакое постороннее вмешательство недопустимо. Например, в большинстве системных функций RTOS в начале кода прерывания запрещаются, а в конце — разрешаются. Это сделано для того, чтобы в процессе изменения внутренних системных данных не могло произойти переключение задачи, что привело бы к ошибкам. Однако этой возможностью не следует злоупотреблять, и крайне желательно, чтобы прерывания не были запрещены в течение длительного времени, поскольку это существенно снижает время реакции системы на внешние события.

Важным моментом является также то, что внутри процедур обработки прерываний невозможен вызов функций ожидания. При попытке вызвать такую функцию будет возвращен код ошибки. Посылка же сигналов и сообщений, а также освобождение семафоров внутри обработчиков прерываний допустимо, хотя и требует некоторых дополнительных действий, а именно: в начале процедуры обработки необходимо сохранить контекст

текущей задачи путем вызова соответствующей системной функции, а в конце вместо обычной инструкции возврата из прерывания (RETI) нужно выполнить вызов системной функции возврата, присутствующей в RTOS API. Это всё требуется для того, чтобы обеспечить корректную обработку прерывания. В обычной ситуации вызов функции отправки сообщения может вызвать переключение на более приоритетную задачу. В случае же обработки прерывания такое поведение недопустимо, и поэтому вместо немедленного переключения функция отправки сообщения просто выставляет флаг переключения. После того как обработка прерывания будет завершена, можно выполнять переключение задач, что и делает системная функция возврата из прерывания, если обнаруживает, что флаг выставлен (именно для этого в начале требуется сохранить контекст задачи).

3. Поддержка MicroDSP-RTOS в MetaDSP

Система MicroDSP-RTOS разрабатывалась для создания проектов в интегрированной среде кросс-разработки MetaDSP, в которой были добавлены новые возможности, облегчающие создание и отладку RTOS-проектов. В первую очередь это система *RTOS Illuminator*, позволяющая просматривать и изменять состояние задач в процессе выполнения программы. Также были добавлены два новых типа профилировки и новый тип проекта, содержащий базовый шаблон для RTOS-проекта. Рассмотрим эти нововведения подробнее.

3.1. RTOS Illuminator

Этот инструмент представляет собой встроенную утилиту для наблюдения за состоянием процессов и управления ими извне. Визуально RTOS Illuminator является присоединяемым окном в среде MetaDSP, в котором на нескольких вкладках отображено состояние подключённых на данный момент задач.

1. На Рис. 2 показана вкладка **Tasks.**

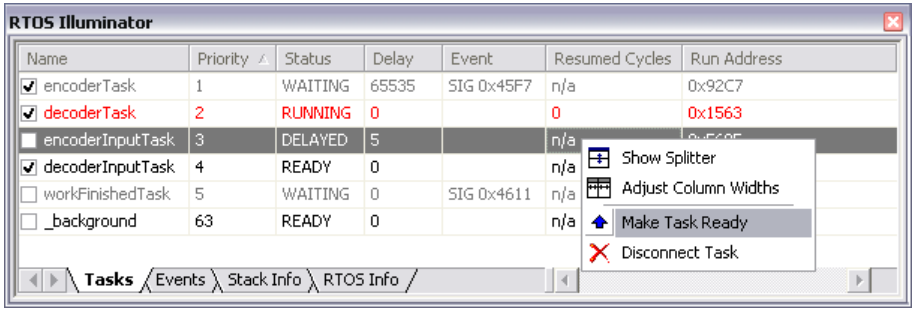

Рис. 2. Вид окна RTOS Illuminator, управление задачами

Она позволяет просматривать текущее состояние процессов: имя процесса (имя подключаемой функции, поле **Name**), приоритет (поле **Priority**), текущий

статус процесса (**Status**). Если задача приостановлена или находится в состоянии ожидания, для неё отображается значение таймаута (**Delay**), а для задач, ожидающих наступления некоторого события, дополнительно указывается, какое именно это событие (**Event**). Для выполняемой в данный момент задачи также указываются количество тактов, прошедшее с момента последнего переключения на эту задачу (**Resumed Cycles**), и текущий адрес выполнения (**Run Address**). Для неактивных задач в поле **Run Address** выводится адрес программной памяти, с которого будет продолжено выполнение задачи. Двойным щелчком мыши по этому полю можно перейти к тому месту исходного кода, которое соответствует указанному адресу, т. е. по сути, к той точке выполнения, в которой задача была прервана.

Помимо этого в этой вкладке можно изменять состояние задач, а именно:

а) отключить задачу (команда **Disconnect Task** в системном меню);
б) перевести задачу из режима ожидания, блокировки или приостановленного состояния в режим готовности (команда **Make Task Ready**);
в) изменить приоритет задачи (редактированием значения в поле **Priority**);
г) изменить значение таймаута (при выставлении таймаута в 0 задача, находящаяся в приостановленном состоянии будет переведена в режим готовности, а для задач, ожидающих наступления какого-либо события значение 0 будет означать бесконечное время ожидания);

Слева от имени каждой задачи присутствует флажок, включив который, можно установить точку останова, срабатывающую в момент переключения RTOS на эту задачу. Эта функция значительно расширяет возможности отладки многопоточных приложений.

2. На Рис. 3 показана вкладка **Events**.

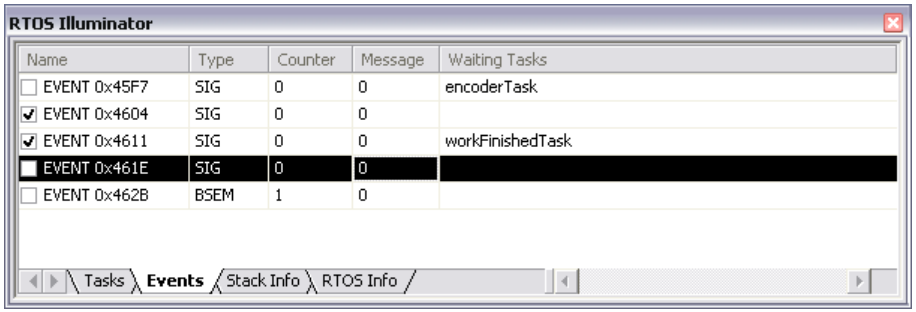

Рис. 3. Вид окна RTOS Illuminator, управление объектами синхронизации

На этой вкладке отображаются все объекты межзадачного взаимодействия и синхронизации, созданные программой. Для каждого объекта выводятся его адрес (в поле **Name**), тип объекта (сигнал, семафор, почтовый ящик и

т. п., поле **Type**) и список задач, ожидающих данный объект синхронизации (поле **Waiting Tasks**). Для сигналов и семафоров дополнительно выводится счётчик, представляющий собой текущее состояние объекта (**Counter**), а для почтовых ящиков и очередей сообщений — указатель на сообщение, если оно присутствует (**Message**).

Так же, как и во вкладке **Tasks**, слева от каждого объекта присутствует флажок, который включает/отключает точку останова, выполняющуюся при наступлении отмеченного события.

3. Вкладка **Stack Info** предоставляет информацию о текущем состоянии стека для каждой задачи. Для текущей задачи это будет просто значение стековых регистров, а для всех остальных задач выводятся значения, сохранённые в контексте при переключении. На этой вкладке также отображаются размер стека, процентное соотношение его использования и максимальный процент использования, который был за всё время работы данной задачи.
4. Вкладка **RTOS Info** отображает сведения о системе RTOS в целом: количество тактов, прошедшее с момента последнего срабатывания таймера, длительность кванта времени, общее число квантов времени, прошедшее с момента старта системы, и версию RTOS.

3.2. RTOS Profiler

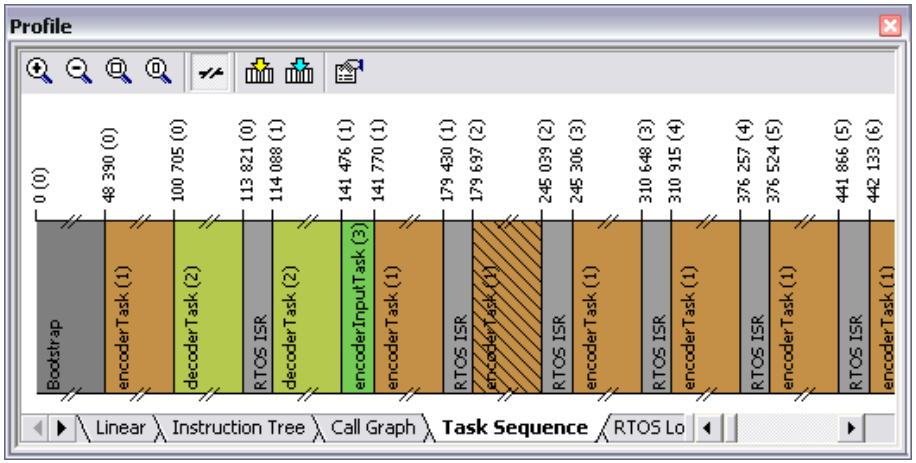

Рис. 4. Вид окна RTOS Profiler, последовательность задач

Изначально, до разработки MicroDSP-RTOS, в MetaDSP присутствовал встроенный профилировщик, предоставляющий информацию о распределении процессорного времени между различными функциями внутри программы, а также собирающий статистику по количеству выполненных

процессорных инструкций разного типа. С появлением MicroDSP-RTOS были добавлены два новых типа профилировки.

1. Отображение последовательности выполняющихся задач (Рис. 4).

   Эта вкладка окна профилировщика предоставляет в наглядном графическом виде, какие задачи и в течение какого промежутка времени выполнялись. Промежуток времени указывается как в процессорных тактах, так и в системных квантах времени.
2. Распределение процессорного времени по задачам (Рис. 5).

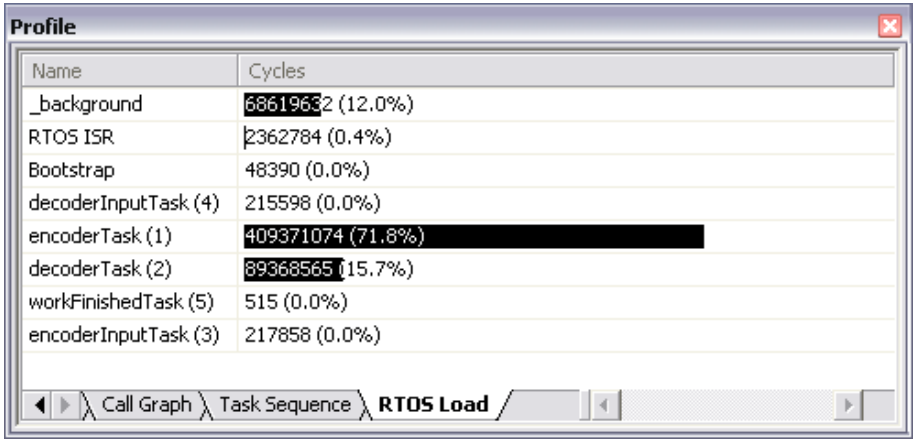

Рис. 5. Вид окна RTOS Profiler, распределение времени по задачам

В этой вкладке окна профилировщика отображается, какую часть процессорного времени занимала каждая задача. Опционально можно также отобразить суммарное время выполнения для системной фоновой задачи (**background**), для процедуры обработки таймерного прерывания (**RTOS ISR** — Interrupt Service Routine), по которому RTOS выполняет переключение задач, и процедуры начальной инициализации (**Bootstrap**).

RTOS-профилировщик позволяет оценить различные показатели разрабатываемого приложения, связанные с мультизадачностью, например, насколько правильно выбран размер кванта времени, выяснить, в течение какого времени система находилась в простое и так далее. Определение значений этих характеристик даёт возможность сравнивать эффективность системы при различных значениях её параметров, что в свою очередь облегчает создание эффективного продукта.

4. Заключение

В настоящей работе была рассмотрена операционная система реального времени MicroDSP-RTOS, разработанная в ИСП РАН для одного из

индустриальных партнеров. Данная система предназначена для обеспечения работы многозадачных решений на базе «системы на чипе» с архитектурой MicroDSP. Реализация MicroDSP-RTOS выполнена полностью на языке ассемблера указанного микропроцессора с предоставлением прикладных интерфейсов для программ на языке C. Были рассмотрены основные возможности системы, этапы её развития, особенности поддержки отладки многозадачных приложений в интегрированной среде кросс-разработки.

Разработанная система имеет следующие характеристики (для времени выполнения указывается максимально возможное время; для перевода в микросекунды рассматривается процессор с частотой 200 МГц):

размер ядра	829 слов
полный размер системы (включая опциональные модули)	1957 слов
время сохранения/восстановления контекста	65 тактов (0,33 мкс)
длительность ISR (8 задач)	474 такта (2,37 мкс)
длительность ISR (63 задачи)	2290 тактов (11,5 мкс)

К настоящему моменту работа над MicroDSP-RTOS завершена, результаты внедрены в производство заказчика; в частности, известно о сотовом телефоне, в котором используется данная система.

Литература

[1] С. Сорокин. Как много ОС РВ хороших… Современные технологии автоматизации, 2/1997, стр. 7–11 (http://www.cta.ru/pdf/1997-2/software1_1997_2.pdf)

[2] С. Сорокин. Windows. Современные технологии автоматизации, 2/1997, стр. 18–20 (http://www.cta.ru/pdf/1997-2/software5_1997_2.pdf)

[3] С. Сорокин. Системы реального времени. Современные технологии автоматизации, 2/1997, стр. 22–29 (http://www.cta.ru/pdf/1997-2/software6_1997_2.pdf)

[4] Comparison between QNX RTOS V6.1, VxWorks AE 1.1 and Windows CE .NET. Dedicated Systems Experts, http://www.dedicated-systems.com.

[5] А. Жданов. Операционные системы реального времени. PCWeek, 8/1999 (http://www.asutp.ru/?p=600591).

[6] А. Жданов, А. Латыев. Замечания о выборе операционных систем при построении систем реального времени. PCWeek, 1/2001 (http://www.asutp.ru/?p=600493)

[7] А. А. Жданов. Что день грядущий нам готовит? (В связи с появлением Windows NT на рынке ОСРВ). http://www.asutp.ru/?p=600308

[8] А. А. Жданов. Современный взгляд на ОС реального времени, (http://www.asutp.ru/?p=600354)

[9] В. Семенюк. Системы реального времени. http://embedded.ifmo.ru/lib/DOC/REFERATS/HTMLRTOS/rtos97.htm.

[10] T. Samuelsson, M. Åkerholm, Department of Computer Science and Engineering; P. Nygren, J. Stärner, L. Lindh. A Comparison of Multiprocessor RTOS Implemented in Hardware and Software. Computer Architecture Laboratory, Mälardalen University, Västerås, Sweden.